



Who Win

เกมชิงไหวพริบประกอบด้วยผู้เล่น 2 คน เริ่มต้นเกมโดยกำหนดจำนวนเต็ม  $n$  จำนวน  $(x_1\ x_2\ \dots\ x_n)$  ที่สามารถซ้ำกันได้ ผู้เล่นแต่ละคนจะผลัดกันเลือกจำนวนเต็มมา 2 จำนวน ที่ไม่ซ้ำกัน  $(x_i$  และ  $x_j)$  แล้วทำการแทนที่  $x_i$  ด้วย  $x_j$  หรือแทนที่  $x_j$  ด้วย  $x_i$  เกมจะหยุดก็ต่อเมื่อจำนวนเต็มทุกตัวมีค่าเท่ากันทั้งหมด และผู้ชนะคือคนที่ทำให้เกมหยุด

จงเขียนโปรแกรมเพื่อหาว่าผู้เล่นคนใดจะเป็นผู้ชนะ โดยกำหนดให้ผู้เล่น 1 เป็นผู้เริ่มเล่นก่อนเสมอ

Input: บรรทัดที่ 1 คือ จำนวนเต็ม  $n$  โดยที่  $2 \leq n \leq 1,000,000$

บรรทัดที่ 2 คือ  $x_1\ x_2\ \dots\ x_n$

Output: ผู้ชนะ “1” หรือ “2”

หมายเหตุ เว้นวรรคข้อมูลแต่ละตัว

Sample :

Input	Output	Notes
6 1 3 2 3 2 1	2	ผู้เล่น 1: 1 -> 3 จะได้ 1 1 2 1 2 1 ผู้เล่น 2: 1 -> 2 จะได้ 1 1 1 1 1 1
6 5 5 2 2 2 2	1	ผู้เล่น 1: 5 -> 2 จะได้ 5 5 5 5 5 5

### Equal Adding

จากจำนวนเต็มบวกที่กำหนดให้  $n$  จำนวน ( $x_1, x_2, \dots, x_n$ ) โดยที่  $n \geq 4$  และ  $x_1 \neq x_2 \neq \dots \neq x_n$  จงเขียนโปรแกรมเพื่อ  
หาค่า  $x$  สี่จำนวนที่ทำให้

$$x_i + x_j = x_k + x_l$$

เมื่อ  $1 \leq i, j, k, l \leq n$

**Input:** บรรทัดที่ 1 คือ  $n$

บรรทัดที่ 2 คือ  $x_1 \ x_2 \ \dots \ x_n$

**Output:**  $x_i + x_j$  หรือ  $x_k + x_l$  หรือในกรณีที่ไม่มีพบ 4 จำนวนดังกล่าว ให้แสดงผลเป็น “not found”

**หมายเหตุ** เว้นวรรคข้อมูลแต่ละตัว

**Sample :**

Input	Output
7 3 7 4 2 1 8 9	11
8 66 30 1 9 8 90 7 55	not found

binaryTree\_1

จงแปลง Binary Tree จากข้อมูลในอาร์เรย์ เป็น Tree แล้วแสดงผลลัพธ์ตาม Preorder Traversal และ post-order Traversal

- Input:** บรรทัดที่ 1 คือ ขนาดของข้อมูลในอาร์เรย์ที่ใช้เก็บข้อมูลของ Binary Tree  
          บรรทัดที่ 2 คือ ข้อมูลของ node ในอาร์เรย์เป็นจำนวนเต็มบวก
- Output:** บรรทัดที่ 1 ข้อมูลจาก Binary Tree ตาม Preorder Traversal  
          บรรทัดที่ 2 ข้อมูลจาก Binary Tree ตาม Post-order Traversal

**หมายเหตุ** เว้นวรรคข้อมูลแต่ละตัว

**Sample :**

Input	Output
5	1 2 4 5 3
1 2 3 4 5	4 5 2 3 1
10	1 2 4 8 9 5 10 3 6 7
1 2 3 4 5 6 7 8 9 10	8 9 4 10 5 2 6 7 3 1

## ✓ BF\_traversal\_Q

จงแปลง Binary Tree จากข้อมูลในอาร์เรย์ เป็น Tree แล้วแสดงผลลัพธ์ตาม Breadth First Traversal โดยใช้ Queue

**Input:** บรรทัดที่ 1 คือ ขนาดของข้อมูลในอาร์เรย์ที่ใช้เก็บข้อมูลของ Binary Tree

บรรทัดที่ 2 คือ ข้อมูลของ node ในอาร์เรย์เป็นจำนวนเต็มบวก

**Output:** ข้อมูลจาก Binary Tree ตาม Breadth First Traversal

**หมายเหตุ** เว้นวรรคข้อมูลแต่ละตัว

**Sample :**

Input	Output
5	1 2 3 4 5
1 2 3 4 5	

## BST\_1

จงสร้าง Binary Search Tree (BST) จากข้อมูลที่ให้ (ข้อมูลไม่เรียงลำดับ) แล้วแสดงผลลัพธ์ตาม Inorder Traversal

**Input:** บรรทัดที่ 1 คือ จำนวนข้อมูล (n)

บรรทัดที่ 2 คือ จำนวนเต็มบวก n จำนวน

**Output:** ข้อมูลจาก Binary Search Tree ตาม Inorder Traversal

**หมายเหตุ** เว้นวรรคข้อมูลแต่ละตัว

**Sample :**

Input	Output
5 5 3 2 4 7	2 3 4 5 7
7 6 5 9 10 8 3 4	3 4 5 6 8 9 10

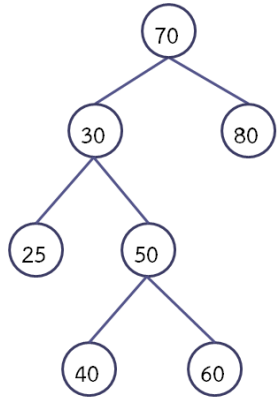
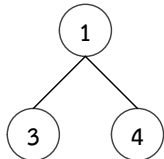
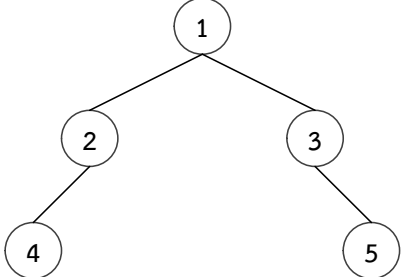
## Leave Node

จาก Binary Tree ที่กำหนดให้ จงเขียนโปรแกรมแสดงค่าข้อมูลทั้งหมดของโหนดใบ โดยเรียงลำดับข้อมูลจากโหนดใบทางซ้ายไปขวา

**Input :** Binary Tree ที่เก็บจำนวนเต็มบวก และอยู่ในรูปของอาร์เรย์  
บรรทัดที่ 1 คือ ขนาดของอาร์เรย์ที่ใช้เก็บ Binary Tree  
บรรทัดที่สอง คือ ข้อมูลแต่ละตัวจากอาร์เรย์ (เว้นวรรคข้อมูลแต่ละตัว)  
หากข้อมูลมีค่าเท่ากับ -1 หมายถึงไม่มีโหนด ณ ตำแหน่งนั้น

**Output :** ข้อมูลของโหนดใบ โดยเรียงลำดับข้อมูลจากโหนดทางซ้ายไปขวา

**Sample :**

Input	Output	Note
11 70 30 80 25 50 -1 -1 -1 -1 40 60	25 40 60 80	
3 1 3 4	3 4	
7 1 2 3 4 -1 -1 5	4 5	

## ✓ Expression Tree

Expression Tree คือ Binary Tree ที่ใช้เก็บนิพจน์ทางคณิตศาสตร์ โหนดของ Expression Tree ประกอบด้วยข้อมูล 2 แบบ คือ

1. Operand คือ ตัวแปร หรือค่าคงที่ที่ถูกดำเนินการทางคณิตศาสตร์ เช่น a, b, x, y หรือ 1, 2, 10, 50 เป็นต้น โดย Operand จะถูกเก็บไว้ที่โหนดใบ
2. Operator คือ เครื่องหมายที่ใช้คำนวณ เช่น +, -, \*, / เป็นต้น โดย Operator จะถูกเก็บไว้ที่โหนดภายใน

ข้อสังเกต Sub Tree เป็นนิพจน์ย่อยที่มี Root Node หรือ Parent Node เป็น Operator เสมอ ตัวอย่างดังรูป

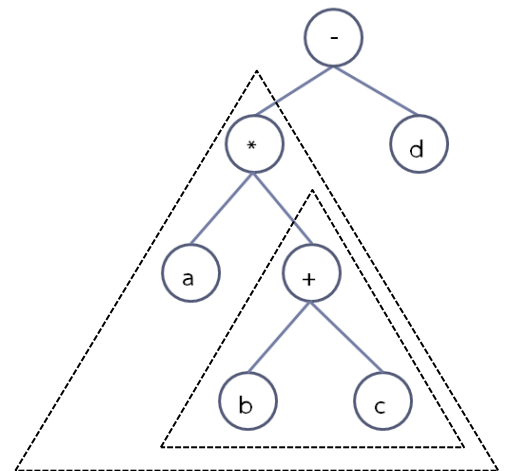
จากรูป นิพจน์ทางคณิตศาสตร์ คือ  $a * (b + c) - d$

ตัวอย่างนิพจน์ย่อย เช่น  $b + c$  และ  $a * (b + c)$  (ในกรอบ

สามเหลี่ยม)

จงเขียนโปรแกรมเพื่อหาผลลัพธ์ของนิพจน์ทางคณิตศาสตร์จาก

Expression Tree ที่กำหนดให้



**Input :** บรรทัดที่ 1 คือ ขนาดของอาร์เรย์ที่ใช้เก็บ Expression Tree

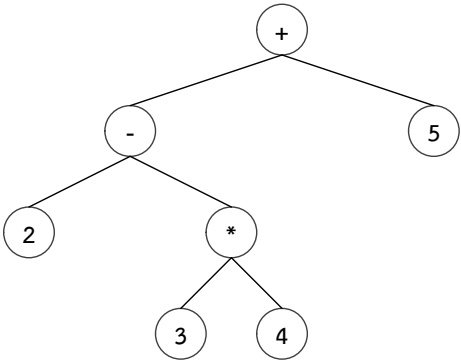
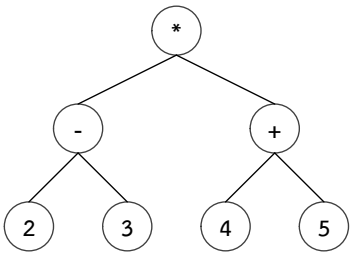
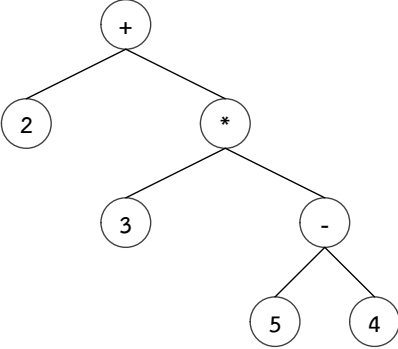
บรรทัดที่สอง คือ ข้อมูลแต่ละตัวจากอาร์เรย์ (เว้นวรรคข้อมูลแต่ละตัว) โดยในที่นี้กำหนดให้

- Operator ประกอบด้วยเครื่องหมาย +, -, \* และ / เท่านั้น
- Operand คือ จำนวนเต็มบวก
- หากข้อมูลมีค่าเท่ากับ -1 หมายถึงไม่มีโหนด ณ ตำแหน่งนั้น จึงไม่ต้องนำมาใช้คำนวณ

ตัวอย่างเช่น [-, \*, d, a, +, -1, -1, -1, -1, b, c] คือ อาร์เรย์ที่ใช้เก็บ Expression Tree ของรูปข้างต้น

**Output :** ผลลัพธ์ที่ได้จากการคำนวณของ Expression Tree (ทศนิยม 2 ตำแหน่ง)

Sample :

Input	Output	Note
11 + - 5 2 * -1 -1 -1 -1 3 4	-5	
7 * - + 2 3 4 5	-9	
15 + 2 * -1 -1 3 - -1 -1 -1 -1 -1 5 4	5	



## ✓ Maximum Path Sum of Binary Tree

จงเขียนโปรแกรมเพื่อหาผลรวมที่มีค่ามากที่สุดของเส้นทางในไบนารีทรี

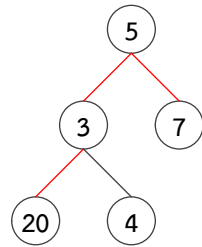
**Input:** Binary Tree ที่เก็บจำนวนเต็ม และอยู่ในรูปของอาร์เรย์

บรรทัดที่ 1 คือ ขนาดของอาร์เรย์ที่ใช้เก็บ Binary Tree

บรรทัดที่ 2 คือ ข้อมูลแต่ละตัวจากอาร์เรย์เป็นจำนวนเต็มบวกหรือจำนวนเต็มลบ (เว้นวรรคข้อมูลแต่ละตัว) หากข้อมูลมีค่าเท่ากับ 0 หมายถึง ไม่มีโหนด ณ ตำแหน่งนั้น

**Output:** ผลรวมที่มีค่ามากที่สุดของเส้นทางในไบนารีทรี

**Example:**

Input	Output	Note
5 5 3 7 20 4	35	
9 -6 2 -9 13 0 8 1 0 -4	15	